Trabalho Final RNA 2020/2

Aluno: Vítor Gabriel Reis Caitité **Matrícula:** 2016111849

1 Introdução

1.1 Objetivo

O objetivo do trabalho é aplicar os modelos de redes neurais artificiais estudados na disciplina a um problema prático referente a um conjunto de dados e que envolve o reconhecimento de sotaques por falantes da língua inglesa. Para facilitar o problema e utilizarmos apenas os métodos aprendidos até aqui, o trabalho considera apenas duas classes, EUA e não-EUA (ou seja, se trata de um problema de classificação binário).

1.2 Dados

Basicamente o arquivo contendo os dados de treinamento contêm 1 coluna de Id (apenas um identificador) seguida de 19 colunas contendo os vetores de entrada x (19 coeficientes espectrais da fala), e por fim, uma última coluna contendo a classificação da língua correspondente, o rótulo y daquela amostra. Foram disponibilizados 3176 dados para treinamento.

Além disso, foi disponibilizado um conjunto de teste (é sobre esse conjunto que os resultados foram obtidos e enviados para avaliação). Assim como o arquivo de treino, o arquivo contendo os dados a serem testados contém um identificador e mais 19 colunas correspondendo as características de entrada extraídas. Foram disponibilizados 1361 dados para teste.

2 Metodologia e Resultados

2.1 Pre-processamento

Antes de se partir de vez para resolução do problema de classificação de fato, foi realizado um escalonamento dos dados. O objetivo disso é alterar os valores das colunas numéricas no conjunto de dados para uma escala comum, aumentando a coesão dos tipos de entrada, sem distorcer as diferenças nos intervalos de valores. Esse processo pode gerar um impacto significativo no modelo.

2.2 Perceptron Simples

Como se sabe, o perceptron simples pode ser utilizado para dividir duas classes linearmente separáveis. Ou seja, o modelo de classificador desenvolvido com esse perceptron de camada única (cuja a rede está mostrada na Figura 1) é na verdade um classificador linear. Dependendo da dimensão do problema esse classificador representa uma reta, um plano ou um hiperplano no espaço de entrada. Como estamos lidando com um problema com 19 variáveis de entrada, então o classificador gerado a partir do perceptron simples representará um hiperplano nesse espaço de entrada, tentando separar os dados linearmente.

Dito isso, é preciso reconhecer que apesar de estar aplicando primeiramente esse método, caso o problema apresente classes não linearmente separáveis, então a classificação feita apresentará um erro alto.

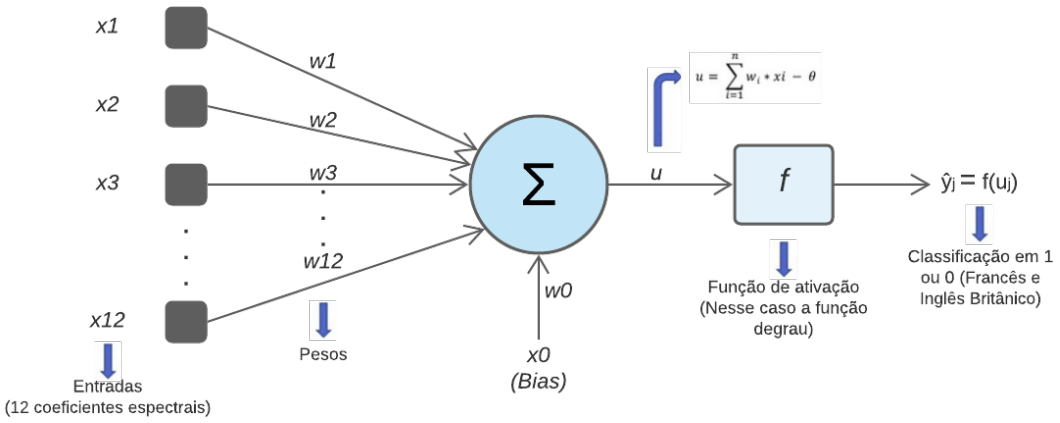

Figura 1: Rede Perceptron Simples. Fonte: Imagem produzida pelo autor

Utilizando - se o algoritmo de treinamento do perceptron simples apresentado durante a disciplina pôde-se, então, treinar o modelo. Esse algoritmo de treinamento baseia-se na aplicação da Equação 1 sobre os dados de treinamento até que o erro global atinja um critério de parada ou complete-se o número máximo de iterações passado como parâmetro.

$$w(t+1) = w(t) + \eta * e(t) * x(t) \tag{1}$$

onde:

- w(t) - valores d vetor de pesos no instante t;
- e(t) - valor do erro no instante t;
- x(t) - vetor de entradas no instante t;
- η - passo de treinamento.

A rede foi então treinada (invocou-se a função de treinamento, passando como parâmetros um passo de 0.1, uma tolerância de 0.01 e um máximo de épocas igual a 10000). Para a validação do modelo, utilizando a biblioteca caret, separou-se os dados aleatoriamente em 70% para treinamento e 30% para validação. Realizando essa validação obteve-se uma acurácia de aproximadamente de 54%, o que já era esperado tendo em vista que pela complexidade do problema ele dificilmente seria linearmente separável.

De toda forma, aplicou-se esse modelo para a classificação dos dados de teste e submeteu-se o resultado no *Kaggle*, obtendo-se uma acurácia de 0.52352 de acordo com o leaderboard do *Kaggle* [1].

2.3 Máquinas de Aprendizado Extremo - ELM

Esse próximo classificador que foi desenvolvido e testado segue o modelo ELM. O algoritmo da ELM nada mais é do que uma maneira diferente de treinar uma rede neural de apenas uma camada oculta. O princípio de funcionamento da ELM é o mesmo de uma RNA, todavia a metodologia de treinamento de uma ELM não é baseada em gradiente descendente. O treinamento da ELM é bastante simples e evita-se gasto computacional com métodos iterativos. Os pesos de entrada e o bias da camada escondida são

[1] This leaderboard is calculated with approximately 50% of the test data. The final results will be based on the other 50%, so the final standings may be different.

escolhidos aleatoriamente. E os pesos da camada de saída são determinados analiticamente (sem ajuste iterativo de parâmetros). O princípio básico da ELM é que a matriz de pesos da camada escondida, selecionada aleatoriamente, seja suficientemente grande para garantir a separabilidade. Assim, garantida uma projeção linearmente separável, pode-se encontrar um separador linear de maneira analítica através da pseudo-inversa.

Para decidir-se o número de neurônios da camada escondida, separou-se os dados de treinamento em dados que realmente foram usados para treinamento (70%) e em dados que foram utilizados para validação (30%). Esses dados são selecionados aleatoriamente utilizando a função *createDataPartition()* do pacote *caret*. Assim, rodou-se o algoritmo, validando-o para diferentes valores de neurônios e percebeu-se que com 240 neurônios na camada escondida obteve-se maior acurácia, com um valor médio de aproximadamente 65% (Foram realizadas 15 execuções diferentes para cada número de neurônios testados). Então. reagrupou-se os dados novamente em um conjunto de treinamento, treinou-se o modelo e aplicou-se aos dados de teste. Ao submeter os resultados obteve-se uma acurácia de 0.66911% de acordo com o leaderboard do *Kaggle*.

2.4 Redes RBF

O próximo classificador testado é baseado no modelo de redes RBF (*Radial Basis Functions Neural Networks*). Esse tipo de rede é caracterizada pela utilização de funções radiais nos neurônios da camada escondida, cujas as respostas são combinadas de maneira linear para gerar a saída.

Utilizou-se um algoritmo de treinamento de uma rede RBF com centros e raios selecionados a partir do algoritmo *K-means* (método de *Clustering* que visa particionar n observações dentre k grupos onde cada observação pertence ao grupo mais próximo da média).

Utilizando a mesma estratégia de validação explicada na subseção anterior encontrou-se que o melhor resultado com esse método foi obtido utilizando de 2200 neurônios na camada escondida (ou seja, aplicando o algoritmo *K-means* com 2200 *clusters*). Com esse modelo conseguiu uma acurácia de validação significativamente melhor que as alcançadas anteriormente, cerca de 75%. Decidiu-se então gerar um modelo utilizando todos os dados de treinamento fornecidos e realizar a submissão no kaggle. Ao se fazer isso, obteve-se uma acurácia de 0.77647% de acordo com o leaderboard do *Kaggle*.

OBS: À título de curiosidade testou-se também um classificados baseado em uma RBF com centros selecionados aleatoriamente (como feito na lista 8), porém esse modelo obteve piores resultados com acurácias de validação entre 60% e 65%.

2.5 Multilayer Perceptron (MLPs)

Uma rede MLP consiste em pelo menos três camadas de nós: uma camada de entrada, uma camada oculta e uma camada de saída. Exceto para os nós de entrada, cada nó é um neurônio que possui uma função de ativação definida. Para treinamento, a MLP utiliza uma técnica de aprendizado supervisionado chamada *backpropagation* [5] [6]. Com o auxílio das múltiplas camadas e ativação não linear, uma rede MLP é capaz de distinguir dados que não são linearmente separáveis. Neste trabalho utilizamos apenas um neurônio de saída (pois estamos lidando com um problema de classificação binário). Porém é importante citar que diferentemente do Perceptron Simples e Adaline, onde existe apenas um único neurônio de saída y, a MLP pode relacionar o conhecimento a vários neurônios de saída [4].

2.5.1 Implementação Própria

Utilizando o conhecimento adquirido durante a disciplina e se baseando nos códigos fornecidos em [2], foi possível desenvolver em R tanto uma função que realiza o treinamento de uma rede MLP com 1 camada

escondida (utilizando o algoritmo *backpropagation*), quanto outra função que calcula a saída de uma rede MLP. A função de ativação utilizada para todos os neurônios da rede foi a tangente hiperbólica. Esses códigos, juntamente com todos utilizados durante o trabalho, podem ser vistos em anexo.

Basicamente o algoritmo de *backpropagation* implementado possui a seguinte equação geral de ajuste de pesos:

$$w_{li} = w_{li} + \eta\delta_i x_l + \alpha\Delta w_{li} \tag{2}$$

Pela equação acima pode-se observar que foi implementado o *backpropagation* com o termo de *momentum* (último termo da expressão). Esse termo determina o efeito das mudanças passadas dos pesos na direção atual do movimento no espaço de pesos [1].

Com esse esquema e utilizando os seguintes parâmetros de treinamento: $\eta = 0.1$, $\alpha = 0.1$, $maxIteration = 1000$, foi possível chegar à uma acurácia de validação média de cerca de 64%. *OBS: Os dados foram separados aleatoriamente em 70% para treinamento e 30% para validação.*

2.5.2 Implementação utilizando o pacote RSNNS em R

Utilizando a biblioteca RSNNS, disponível em R, foi possível criar uma rede MLP e treiná-la e testá-la. Os dados de treinamento foram novamente separados entre dados de treino e validação, e com base na acurácia média entre 10 execuções, foi possível avaliar modelos para diferentes configurações.

O modelo que apresentou melhores resultados durante os testes foi o definido no código abaixo:

```
model <- mlp(x_train, y_train, size = 45, maxit = 2000, initFunc = "Randomize_Weights",
    initFuncParams = c(-0.3, 0.3), learnFunc = "Rprop", learnFuncParams = c(0.1, 0.3, 0.1)
    , updateFunc = "Topological_Order", updateFuncParams = 0.1, hiddenActFunc = "Act_TanH"
    , shufflePatterns = TRUE, linOut = FALSE, inputsTest = NULL, targetsTest = NULL)
```

Listing 1: Modelo MLP

Contudo, a acurácia de validação encontrada com esse modelo, ficou em torno de 65%, ainda bem abaixo do encontrado utilizando a técnica de RBF.

No gráfico da Figura 3 é possível acompanhar a evolução do treinamento a cada iteração (esse gráfico mostra a soma do erro quadrático por iteração).

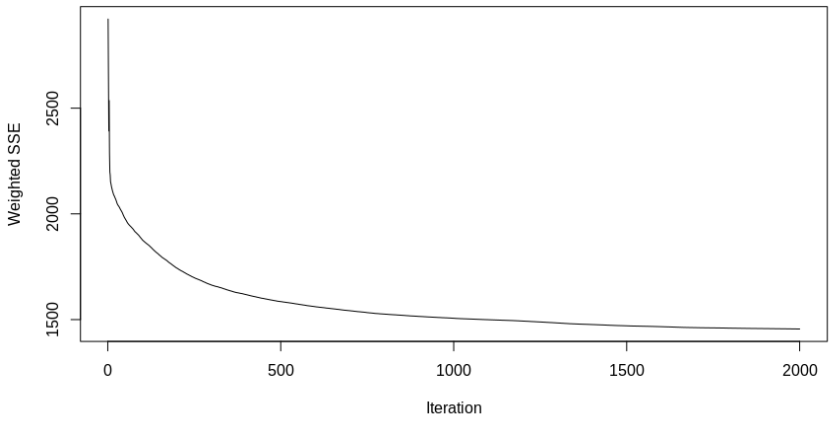

Figura 2: Gráfico Weighted SSE x Iteration.

2.5.3 Implementação utilizando a biblioteca Keras em Python

Keras é uma biblioteca que possibilita desenvolver redes neurais com códigos de alto-nível escritos em Python e roda como frontend em TensorFlow ou Theano. Algumas vantagens em se utilizar essa lib. são:

- Prototipagem rápida e pática (total modularidade, minimalismo e extensibilidade).
- Suporte a diferentes tipos de redes, incluindo combinação delas.
- Possibilita rodar treinamento e modelo tanto na CPU quanto na GPU.
- Assim com o pacote RSNNS do R, keras também possui diversos tipos de funções de ativação, métodos de treinamento e técnicas de validação (dentre elas a k-fold cross validation).

Utilizando então essa biblioteca foi possível criar uma rede MLP, treiná-la e testá-la. Para escolha e validação do modelo foi utilizada a técnica 10-fold Crossvalidation.

A validação cruzada busca avaliar a capacidade de generalização de um modelo, a partir de um conjunto de dados. De acordo com [7], "o método de validação cruzada denominado k-fold consiste em dividir o conjunto total de dados em k subconjuntos mutuamente exclusivos do mesmo tamanho e, a partir daí, um subconjunto é utilizado para teste e os k-1 restantes são utilizados para estimação dos parâmetros, fazendo-se o cálculo da acurácia do modelo. Este processo é realizado k vezes alternando de forma circular o subconjunto de teste".

Utilizando então o 10-fold Crossvalidation pôde-se testar diferentes parâmetros para o modelo. O modelo escolhido (que obteve uma acurácia média de 67%) foi treinado utilizando o otimizador que implementa o algoritmo de Adam. A otimização de Adam é um método de descida gradiente estocástico baseado na estimativa adaptativa de momentos de primeira e segunda ordem [3]. A taxa de aprendizado foi configurada para 0.001, o número de neurônios da camada escondida foi 35 e a função de ativação utilizada para os neurônios foi tangente hiperbólica.

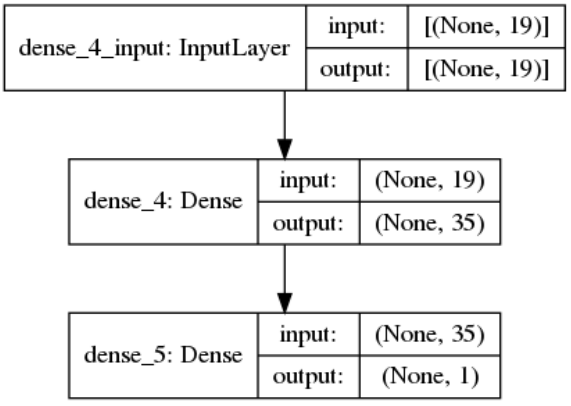

Figura 3: Modelo gerada utilizando a biblioteca Keras em Python.

3 Conclusão

Com esse trabalho foi possível lidar na prática com um conjunto de dados reais, com uma quantidade de entradas elevada. Além disso, possibilitou-nos aplicar todos os conhecimentos adquiridos durante a disciplina. Durante esse TP foram aplicados os seguintes métodos: Perceptron Simples, Redes ELM, Redes RBF e Redes MLP.

Apesar do modelo mais estudado durante o trabalho ter sido as redes MLP, aquele que obteve o melhor resultado foi a rede RBF. Com ela conseguiu-se alcançar uma acurácia de 77% considerando 50% dos dados utilizados para o *leaderboard* do *kaggle*. Isso mostra a importância de se avaliar diferentes técnicas no momento em que se está lidando com um database complexo como esse.

Referências

[1] André Ponce de Leon F. de Carvalho. *Perceptron Multi-Camadas (MLP).* https://sites.icmc.usp.br/andre/research/neural/mlp.htm, Accessed in 26 march 2021.

[2] Antônio de Pádua Braga. *Aprendendo com Exemplos: Princípios de Redes Neurais Artificiais e Reconhecimento de Padrões.*

[3] Keras. *Adam.* https://keras.io/api/optimizers/adam/, Accessed in 26 march 2021.

[4] Sandro Moreira. *Rede Neural Perceptron Multicamadas.* https://medium.com/ensina-ai/rede-neural-perceptron-multicamadas-f9de8471f1a9, Accessed in 26 march 2021.

[5] Frank. Rosenblatt. Principles of neurodynamics: Perceptrons and the theory of brain mechanisms, 1961.

[6] Geoffrey E. Hinton Rumelhart, David E. and R. J. Williams. "learning internal representations by error propagation", 1986.

[7] Wikipédia. *Validação cruzada.* https://pt.wikipedia.org/wiki/Valida%C3%A7%C3%A3o_cruzada, Accessed in 26 march 2021.

4 Anexo - Códigos Utilizados

4.1 Treinamento Perceptron Simples

```
trainPerceptron <- function ( xin , yd , eta , tol , maxepocas , par )
{
  dimxin<-dim( xin )
  N <-dimxin[ 1 ]
  n<-dimxin[ 2 ]
  if ( par==1){
    wt<-as.matrix ( runif(n+1) - 0.5)
    xin<-cbind ( 1 , xin )
  } else {
    wt<-as.matrix ( runif ( n ) - 0.5)
  }
  nepocas<-0
  eepoca<-tol + 1

  evec<-matrix ( nrow =1 , ncol=maxepocas )
  while( ( nepocas < maxepocas ) && ( eepoca>tol ) )
  {
    ei2<-0
    xseq<-sample(N)
    for ( i in 1:N)
    {
      irand<-xseq[i]
      yhati<-1.0 * ( ( xin[irand , ] %*% wt ) >= 0 )
      ei<-yd[irand]- yhati
      dw<-as.vector(eta) * as.vector(ei) * xin[ irand  , ]
      wt<-wt+dw
      ei2<-ei2 + ei * ei
    }
    nepocas<-nepocas+1
    evec[ nepocas ]<-ei2/N

```

```
    eepoca<-evec[nepocas]
  }
  retlist<-list ( wt, evec[ 1:nepocas]
  return (retlist)
}
```

Listing 2: Função de treinamento de um perceptron simples em R

4.2 Resposta do Perceptron Simples

```
yperceptron <- function(xvec, w, par){
  # xvec: vetor de entrada
  # w: vetor de pesos
  # par: se adiciona ou nao o vetor de 1s na entrada
  # yperceptron: resposta do perceptron
  if ( par==1){
    xvec<-cbind ( 1 , xvec )
  }
  u <- xvec %*% w
  y <- 1.0 * (u>=0)
  return(as.matrix(y))
}
```

Listing 3: Função que calcula a resposta de um perceptron simples em R

4.3 Classificador Utilizando o Perceptron Simples

```

rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-recognition/
    Simple_Perceptron/trainPerceptron.R")
source("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-recognition/
    Simple_Perceptron/yperceptron.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/escalonamento_matrix.R")
library(caret)

# Carregando base de dados:
path <- file.path("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-
    recognition/databases", "treino.csv")
data_train <- read.csv(path)
path <- file.path("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-
    recognition/databases", "teste.csv")
data_test <- read.csv(path)

# Separando dados de entrada e sa da e treino e teste:
x_train <- as.matrix(data_train[1:3176, 2:20])
class <- as.matrix(data_train[1:3176, 21])
y_train <- rep(0,nrow(x_train))
for (count in 1:length(class)) {
  if (class[count] == 1 ){
    y_train[count] <- 1
  }
  else{
    y_train[count] <- 0
  }
}
x_test <- as.matrix(data_test[1:1361, 2:20])
```

```
27
28 # Escalonando os valores dos atributos para que fiquem restritos entre 0 e 1
29 x_all <- rbind(x_train, x_test)
30 x_all <- staggeringMatrix(x_all, nrow(x_all), ncol(x_all))
31 x_train <- x_all[1:nrow(x_train),]
32 x_test <- x_all[(nrow(x_train)+1):(nrow(x_train)+nrow(x_test)),]
33
34 # Treinando modelo:
35 retlist<-trainPerceptron(x_train, y_train, 0.2, 0.1, 10000, 1)
36 W<-retlist[[1]]
37
38 # Calculando acuracia de treinamento
39 length_train <- length(y_train)
40 y_hat_train <- as.matrix(yperceptron(x_train, W, 1), nrow = length_train, ncol = 1)
41 accuracy_train <- 1-((t(y_hat_train-y_train) %*% (y_hat_train-y_train))/length_train)
42
43 # Rodando dados de teste:
44 y_hat_test <- as.matrix(yperceptron(x_test, W, 1), nrow = length_test, ncol = 1)
45 y <- ifelse(y_hat_test == 0, -1, 1)
46
47 Id <- 3177:4537
48 table <- data.frame(Id, y)
49 write.csv(table, "prediction_perceptron.csv", row.names = FALSE)
```

Listing 4: Classificador para o problema proposto, utilizando um perceptron simples em R

4.4 Treinamento de ELMs

```
1  library("corpcor")
2
3  trainELM <- function(xin, yin, p, par){
4    n <- dim(xin)[2] # Dimensao da entrada
5
6    #Adiciona ou nao o termo de polarizacao
7    if(par == 1){
8      xin<-cbind(1,xin)
9      Z<-replicate(p, runif(n+1, -0.5, 0.5))
10   }
11   else{
12     Z<-replicate(p, runif(n, -0.5, 0.5))
13   }
14   H<-tanh(xin %*% Z)
15
16   W<-pseudoinverse(H)%*%yin
17   #W<-(solve(t(H) %*% H) %*% t(H)) %*% yin
18
19   return(list(W,H,Z))
20 }
```

Listing 5: Função de treinamento de ELMs em R

4.5 Resposta da Rede ELM

```
1 YELM<-function(xin, Z, W, par){
2   n<-dim(xin)[2]
3
4   # Adiciona ou nao termo de polarizacao
```

```
  if(par == 1) {
    xin<-cbind(1, xin)
  }
  H<-tanh(xin%*%Z)
  y_hat<-sign(H %*% W)
  return(y_hat)
}
```

Listing 6: Função que calcula a resposta de uma rede ELM em R

4.6 Classificador Utilizando uma Rede ELM

```
rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-recognition/ELM/trainELM.R")
source("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-recognition/ELM/YELM.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/escalonamento_matrix.R")
library(caret)

# Carregando base de dados:
path <- file.path("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-recognition/databases", "treino.csv")
data_train <- read.csv(path)
path <- file.path("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-recognition/databases", "teste.csv")
data_test <- read.csv(path)


# Separando dados de entrada e sa da e treino e teste:
x_train <- as.matrix(data_train[1:3176, 2:20])
y_train <- as.matrix(data_train[1:3176, 21])
x_test <- as.matrix(data_test[1:1361, 2:20])

# Escalonando os valores dos atributos para que fiquem restritos entre 0 e 1
x_all <- rbind(x_train, x_test)
x_all <- staggeringMatrix(x_all, nrow(x_all), ncol(x_all))
x_train <- x_all[1:nrow(x_train), ]
x_test <- x_all[(nrow(x_train)+1):(nrow(x_train)+nrow(x_test)), ]

p <- 240 # n mero de neuronios
executions <- 1
results <- matrix(nrow = nrow(x_test), ncol = executions)

for (index in 1:executions){
  # Treinando modelo:
  retlist<-trainELM(x_train, y_train, p, 1)
  W<-retlist[[1]]
  H<-retlist[[2]]
  Z<-retlist[[3]]

  # Calculando acuracia de treinamento
  length_train <- length(y_train)
  y_hat_train <- as.matrix(YELM(x_train, Z, W, 1), nrow = length_train, ncol = 1)
  accuracy_train<-((sum(abs(y_hat_train + y_train)))/2)/length_train
  #print(accuracy_train)

  # Rodando dados de teste:
```

```
  y_hat_test <- as.matrix(YELM(x_test, Z, W, 1), nrow = length_test, ncol = 1)
  results[,index] <- y_hat_test
}

y <- rep(0, nrow(y_hat_test))
for (index in 1:nrow(y_hat_test)) {
  if(sum(results[index,] == 1) > (executions/2)){
    y[index] <- 1
  }
  else{
    y[index] <- -1
  }
}

Id <- 3177:4537
table <- data.frame(Id, y)
write.csv(table, "prediction_eml.csv", row.names = FALSE)
```

Listing 7: Classificador para o problema proposto, utilizando uma rede ELM em R

4.7 Validação Rede ELM

```
rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-recognition/ELM/trainELM.R")
source("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-recognition/ELM/YELM.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/escalonamento_matrix.R")
library(caret)

# Carregando base de dados:
path <- file.path("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-recognition/databases", "treino.csv")
data_train <- read.csv(path)

executions <- 15

p <- c(228, 230, 232, 240, 243, 238) # numero de neuronios

for (p in p){
  results <- rep(0, executions)
  for (index in 1:executions){
    # Separando dados de entrada e saida e treino e teste:
    particao <- createDataPartition(1:dim(data_train)[1],p=.7)
    train <- as.matrix(data_train[particao$Resample1,])
    validation <- as.matrix(data_train[-particao$Resample1,])

    x_train <- as.matrix(train[, 2:(ncol(train)-1)])
    y_train <- as.matrix(train[, ncol(train)])
    x_validation <- as.matrix(validation[, 2:(ncol(train)-1)])
    y_validation <- as.matrix(validation[, ncol(train)])

    # Escalonando os valores dos atributos para que fiquem restritos entre 0 e 1
    x_all <- rbind(x_train, x_validation)
    x_all <- staggeringMatrix(x_all, nrow(x_all), ncol(x_all))
    x_train <- x_all[1:nrow(x_train), ]
    x_validation <- x_all[(nrow(x_train)+1):(nrow(x_train)+nrow(x_validation)), ]

```

```
34
35      length_train <- length(y_train)
36      length_validation <- length(y_validation)
37
38      # Treinando modelo:
39      retlist<-trainELM(x_train, y_train, p, 1)
40      W<-retlist[[1]]
41      H<-retlist[[2]]
42      Z<-retlist[[3]]
43
44      # Calculando acuracia de treinamento
45      y_hat_train <- as.matrix(YELM(x_train, Z, W, 1), nrow = length_train, ncol = 1)
46      accuracy_train<-((sum(abs(y_hat_train + y_train)))/2)/length_train
47      #print(paste("Acuracia de treinamento para p = ", p, " e ", accuracy_train))
48
49      # Rodando dados de teste:
50      y_hat_test <- as.matrix(YELM(x_validation, Z, W, 1), nrow = length_validation, ncol =
            1)
51      accuracy_validation<-((sum(abs(y_hat_test + y_validation)))/2)/length_validation
52      results[index] <- accuracy_validation
53    }
54    print(paste("Acuracia de teste para p = ", p, " e ", mean(results)))
55  }
```

Listing 8: Algoritmo para validação da rede ELM

4.8 Treinamento de RBFs

```
1  # Funcao de treinamento de uma rede RBF.
2  library("corpcor")
3
4  trainRBF <- function(xin, yin, p){
5    ######## Funcao radial Gaussiana #########
6    pdfnvar<-function(x,m,K,n){
7      if (n==1) {
8        r<-sqrt(as.numeric(K))
9        px<-(1/(sqrt(2*pi*r*r)))*exp(-0.5 *((x-m)/r)^2)
10     }
11     else {
12       px<-((1/(sqrt((2*pi)^n * (det(K))))) * exp (-0.5 * (t(x-m) %*% (solve(K)) %*% (x-m))
             )) #eq 6.5
13     }
14   }
15   ##############################################
16   N<-dim(xin)[1] # numero de amostras
17   n<-dim(xin)[2] # dimensao de entrada (deve ser maior que 1)
18
19   xin <- as.matrix(xin) # garante que xin seja matriz
20   yin <- as.matrix(yin) # garante que yin seja matriz
21
22   # Aplica o algoritmo kmeans para separar os clusters
23   xclust<-kmeans(xin, p)
24
25   # Armazena vetores de centros das funcoes:
26   m <- as.matrix (xclust$centers)
27   covlist <- list()
28
29   # Estima matrizes de covariancia para todos os centros:
```

```
  for ( i in 1:p)
  {
    ici <- which(xclust$cluster == i )
    xci <- xin [ici , ]
    if(n==1){
      covi <- var(xci)
    }
    else{
      row <- dim(xci)[1];
      if(is.null(row)){
        row <- 0
      }
      # Para garantir que nao havera erro (caso tenha apenas uma linha)
      if(row > 1){
        covi <- cov(xci)
      }
      else{
        # cov de 2 linhas iguais que vai dar 0
        covi <- cov(matrix(c(xci, xci), nrow = 2))
      }
    }
    covlist [[i]] <- covi
  }

  H <- matrix(nrow = N, ncol = p)
  # Calcula matriz H
  for (j in 1:N) {
    for (i in 1:p) {
      mi <- m[i, ]
      covi <- covlist[i]
      covi <- matrix(unlist(covlist[i]), ncol = n, byrow = T) + 0.001 * diag(n)
      H[j,i] <- pdfnvar(xin[j, ] , mi, covi, n)
    }
  }

  Haug <-cbind(1, H)
  W <- pseudoinverse(Haug) %*% yin

  return (list(m, covlist , W, H))
}
```

Listing 9: Função de treinamento da rede RBF em R

4.9 Resposta da Rede RBF

```
# Funcao que calcula a saida de uma rede RBF.
library("corpcor")

YRBF <- function(xin , modRBF){
  ######## Funcao radial Gaussiana ########
  pdfnvar<-function(x,m,K,n){
    if (n==1) {
      r<-sqrt(as.numeric(K))
      px<-(1/(sqrt(2*pi*r*r))) * exp(-0.5 *((x-m)/r)^2)
    }
    else {
      px<-((1/(sqrt((2*pi)^n * (det(K))))) * exp (-0.5 * (t(x-m) %*% (solve(K)) %*% (x-m))
          ))
```

```
    }
  }
  ##########################################
  N <- dim(xin)[1] # numero de amostras
  n <- dim(xin)[2] # dimensao de entrada (deve ser maior que 1)
  m <- as.matrix(modRBF[[1]])
  covlist <- modRBF[[2]]
  p <- length(covlist) # Numero de funcoes radiais
  W <- modRBF [[3]]

  xin <- as.matrix(xin) # garante que xin seja matriz

  H <- matrix(nrow = N, ncol = p)
  # Calcula matriz H
  for (j in 1:N) {
    for (i in 1:p) {
      mi <- m[i, ]
      covi <- covlist[i]
      covi <- matrix(unlist(covlist[i]), ncol = n, byrow = T) + 0.001 * diag(n)
      H[j,i] <- pdfnvar(xin[j, ], mi, covi, n)
    }
  }

  Haug <- cbind(1, H)
  Yhat <- Haug %*% W
  return(Yhat)
}
```

Listing 10: Função que calcula a resposta de uma rede RBF em R

4.10 Validação Rede RBF

```
rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-recognition/RBF/trainRBF.R")
source("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-recognition/RBF/YRBF.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/escalonamento_matrix.R")
library(caret)

# Carregando base de dados:
path <- file.path("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-recognition/databases", "treino.csv")
data_train <- read.csv(path)

executions <- 5

ps <- c(135, 1600, 1900, 2100, 2200) # numero de neuronios

results <- matrix(rep(0, (executions*length(ps))), nrow = executions)
for (index in 1:executions){
  # Separando dados de entrada e saida e treino e teste:
  particao <- createDataPartition(1:dim(data_train)[1],p=.7)
  train <- as.matrix(data_train[particao$Resample1,])
  validation <- as.matrix(data_train[-particao$Resample1,])

  x_train <- as.matrix(train[, 2:(ncol(train)-1)])
  y_train <- as.matrix(train[, ncol(train)])
```

```
  x_validation <- as.matrix(validation[, 2:(ncol(train)-1)])
  y_validation <- as.matrix(validation[, ncol(train)])

  # Escalonando os valores dos atributos para que fiquem restritos entre 0 e 1
  x_all <- rbind(x_train, x_validation)
  x_all <- staggeringMatrix(x_all, nrow(x_all), ncol(x_all))
  x_train <- x_all[1:nrow(x_train), ]
  x_validation <- x_all[(nrow(x_train)+1):(nrow(x_train)+nrow(x_validation)), ]


  length_train <- length(y_train)
  length_validation <- length(y_validation)
  for (p in ps){
    # Treinando modelo:
    modRBF<-trainRBF(x_train, y_train, p)

    # Calculando acur cia de treinamento
    y_hat_train <- as.matrix(YRBF(x_train, modRBF), nrow = length_train, ncol = 1)
    yt <- (1*(y_hat_train >= 0)-0.5)*2
    accuracy_train<-((sum(abs(yt + y_train)))/2)/length_train
    #print(paste("Acuracia de treinamento para p = ", p, "    ", accuracy_train))

    # Rodando dados de teste:
    y_hat_test <- as.matrix(YRBF(x_validation, modRBF), nrow = length_test, ncol = 1)
    yt <- (1*(y_hat_test >= 0)-0.5)*2
    accuracy_validation<-((sum(abs(yt + y_validation)))/2)/length_validation
    results[index, match(p, ps)] <- accuracy_validation
    print(paste("Acuracia de teste para p = ", p, " e ", accuracy_validation))
  }
}
print("
    ----------------------------------------------------------------------------------------------------
    ")

for (p in ps){
  print(paste("Acuracia de teste media para p = ", p, " e ", mean(results[, match(p, ps)])
      ))
}
```

Listing 11: Algoritmo para validação da rede RBF

4.11 Classificador Utilizando uma Rede RBF

```
rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-recognition/
    RBF/trainRBF.R")
source("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-recognition/
    RBF/YRBF.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/escalonamento_matrix.R")
library(caret)

# Carregando base de dados:
path <- file.path("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-
    recognition/databases", "treino.csv")
data_train <- read.csv(path)
path <- file.path("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-
    recognition/databases", "teste.csv")
data_test <- read.csv(path)
```

```r

# Separando dados de entrada e saida e treino e teste:
x_train <- as.matrix(data_train[1:3176, 2:20])
y_train <- as.matrix(data_train[1:3176, 21])
x_test <- as.matrix(data_test[1:1361, 2:20])

# Escalonando os valores dos atributos para que fiquem restritos entre 0 e 1
x_all <- rbind(x_train, x_test)
x_all <- staggeringMatrix(x_all, nrow(x_all), ncol(x_all))
x_train <- x_all[1:nrow(x_train), ]
x_test <- x_all[(nrow(x_train)+1):(nrow(x_train)+nrow(x_test)), ]

p <- 2100 # numero de neurunios
executions <- 1
results <- matrix(nrow = nrow(x_test), ncol = executions)

for (index in 1:executions){
  # Treinando modelo:
  modRBF<-trainRBF(x_train, y_train, p)

  # Calculando acuracia de treinamento
  length_train <- length(y_train)
  y_hat_train <- as.matrix(YRBF(x_train, modRBF), nrow = length_train, ncol = 1)
  yt <- (1*(y_hat_train >= 0)-0.5)*2
  accuracy_train<-((sum(abs(yt + y_train)))/2)/length_train
  #print(accuracy_train)

  # Rodando dados de teste:
  y_hat_test <- as.matrix(YRBF(x_test, modRBF), nrow = length_test, ncol = 1)
  yt <- (1*(y_hat_test >= 0)-0.5)*2
  results[,index] <- yt
}

y <- rep(0, nrow(yt))
for (index in 1:nrow(yt)) {
  if(sum(results[index,] == 1) > (executions/2)){
    y[index] <- 1
  }
  else{
    y[index] <- -1
  }
}

Id <- 3177:4537
table <- data.frame(Id, y)
write.csv(table, "prediction_rbf.csv", row.names = FALSE)
```

Listing 12: Classificador para o problema proposto, utilizando uma rede RBF em R

4.12 Escalonamento dos Dados

```r
# Funcao que recebe uma matriz e suas dimensoes e retorna uma matriz
# de mesma dimensao porem com sua colunas escalonadas.
staggeringMatrix <- function(matrix, rows, columns ){
  staggeredMatrix <- matrix(rep(0, rows*columns), ncol = columns, nrow = rows)
  # Escalonando dados:
  for (j in 1:columns) {
```

```
    for (i in 1:rows) {
      staggeredMatrix[i,j] <- (matrix[i,j] - min(matrix[,j])) / (max(matrix[,j]) - min(
          matrix[,j]))
    }
  }
  return(staggeredMatrix)
}
```

Listing 13: Função para escalonamento dos dados de uma matriz em R

4.13 Backpropagation com termo de *momentum*

```
backpropagation <- function(x_train, y_train, p, tol, eta, alfa, max_epoch){
  sech2<-function(u){
    return(((2/(exp(u)+exp(-u)))*(2/(exp(u)+exp(-u)))))
  }

  #Inicializacao dos pesos.
  n <- ncol(x_train)
  N <- nrow(x_train)
  m <- ncol(y_train)

  #Matriz Z nxp, neste caso (n+1) x p
  Z<-matrix(runif((n+1)*p)-0.5,ncol=p,nrow=n+1)
  Zt <- Z
  Ztless1 <- Z

  #Matriz W pxm, neste caso (p+1) x m
  W<-matrix(runif((p+1)*1)-0.5,ncol=1,nrow=p+1)
  Wt <- W
  Wtless1 <- W

  x_actual <- matrix(nrow=(n+1),ncol=1)

  n_epoch <- 0
  error_epoch <- tol+1
  evec<-matrix(nrow=max_epoch,ncol=1)

  while((n_epoch < max_epoch) && (error_epoch > tol))
  {
    ei2<-0

    #Sequencia aleatoria de treinamento.
    xseq<-sample(N)
    for(i in 1:N)
    {
      #Amostra dado da sequencia aleatoria.
      irand <- xseq[i]
      x_actual[1:n,1] <- x_train[irand,]
      x_actual[n+1,1] <- 1

      y_actual <- y_train[irand, ]

      U<-t(x_actual)%*%Z

      H<-tanh(U)
      Haug<-cbind(H,1)

```

```
        O<-Haug%*%W
        yhat <- tanh(O)

        error <- y_actual-yhat
        flinhaO <-sech2(O)
        dO <- error * flinhaO #Produto elemento a elemento

        Wminus <- W[-(p+1),]  #Saida do bias nao se propaga
        ehidden <- dO%*%t(Wminus)
        flinhaU<-sech2(U)
        dU<-ehidden*flinhaU  #Produto elemento a elemento

        W<-Wt+eta*(t(Haug)%*%dO)+alfa*(Wt - Wtless1)
        Wtless1 <- Wt
        Wt <- W

        Z<-Zt+eta*(x_actual%*%dU)+alfa*(Zt - Ztless1)
        Ztless1 <- Zt
        Zt <- Z

        ei2<-ei2+(error%*%t(error))
      }
      #Incrementa n mero de epocas.
      n_epoch<-n_epoch+1
      evec[n_epoch]<-ei2/N
      #Armazena erro por  poca .
      error_epoch<-evec[n_epoch]
    }
    return(list(W, Z, evec, n_epoch))
}
}
```

Listing 14: Std backpropagation com termo de momento.

4.14 Resposta da Rede MLP

```
YMLP <- function(x_in, modMLP){
  W<-modMLP[[1]]
  Z<-modMLP[[2]]
  u <- cbind(x_in,1) %*% Z
  H<-tanh(u)
  O<-cbind(H,1)%*%W
  yhat_test<-tanh(O)
  return(yhat_test)
}
```

Listing 15: Função que calcula a resposta de uma rede MLP em R

4.15 Validação da rede MLP

```
rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/exemplos/MLP_backpropagation_tanh_momentum.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/YMLP_tanh.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/escalonamento_matrix.R")
library(caret)
library(RSNNS)

```

```r
# Carregando base de dados:
path <- file.path("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-recognition/databases", "treino.csv")
data_train <- read.csv(path)

executions <- 10

ps <- c(10,15,20,25,30,35) # numero de neuronios

results <- matrix(rep(0, (executions*length(ps))), nrow = executions)
for (index in 1:executions){
  # Separando dados de entrada e saida e treino e teste:
  particao <- createDataPartition(1:dim(data_train)[1],p=.7)
  train <- as.matrix(data_train[particao$Resample1,])
  validation <- as.matrix(data_train[-particao$Resample1,])

  x_train <- as.matrix(train[, 2:(ncol(train)-1)])
  y_train <- as.matrix(train[, ncol(train)])
  x_validation <- as.matrix(validation[, 2:(ncol(train)-1)])
  y_validation <- as.matrix(validation[, ncol(train)])

  # Escalonando os valores dos atributos para que fiquem restritos entre 0 e 1
  x_all <- rbind(x_train, x_validation)
  x_all <- normalizeData(x_all, type = "norm")
  x_train <- x_all[1:nrow(x_train), ]
  x_validation <- x_all[(nrow(x_train)+1):(nrow(x_train)+nrow(x_validation)), ]


  length_train <- length(y_train)
  length_validation <- length(y_validation)
  for (p in ps){
    # Treinando modelo:
    modMLP<-backpropagation(x_train, y_train, p, 1, 0.1, 0.1, 5000)

    # Calculando acuracia de treinamento
    y_hat_train <- as.matrix(YMLP(x_train, modMLP), nrow = length_train, ncol = 1)
    yt <- (1*(y_hat_train >= 0)-0.5)*2
    accuracy_train<-((sum(abs(yt + y_train)))/2)/length_train
    #print(paste("Acuracia de treinamento para p = ", p, "    ", accuracy_train))

    # Rodando dados de teste:
    y_hat_test <- as.matrix(YMLP(x_validation, modMLP), nrow = length_test, ncol = 1)
    yt <- (1*(y_hat_test >= 0)-0.5)*2
    accuracy_validation<-((sum(abs(yt + y_validation)))/2)/length_validation
    results[index, match(p, ps)] <- accuracy_validation
    print(paste("Acuracia de teste para p = ", p, "    ", accuracy_validation))
  }
}
print("
    ------------------------------------------------------------------------------------------------------------------------
    ")

for (p in ps){
  print(paste("Acuracia de teste media para p = ", p, "    ", mean(results[, match(p, ps)
    ])))
}
```

Listing 16: Algoritmo para validação da rede MLP em R

4.16 Validação da rede MLP - RSNNS

```
rm(list=ls())
source("~/Documents/UFMG/9/Redes Neurais/exemplos/MLP_backpropagation_tanh_momentum.R")
source("~/Documents/UFMG/9/Redes Neurais/exemplos/YMLP_tanh.R")
library(caret)
library(RSNNS)

# Carregando base de dados:
path <- file.path("~/Documents/UFMG/9/Redes Neurais/TP2/final-work-neural-network-accent-recognition/databases", "treino.csv")
data_train <- read.csv(path)

executions <- 1

ps <- c(15,20,25,35,40,45) # n mero de neur nios

results <- matrix(rep(0, (executions*length(ps))), nrow = executions)
for (index in 1:executions){
  # Separando dados de entrada e sa da e treino e teste:
  particao <- createDataPartition(1:dim(data_train)[1],p=.7)
  train <- as.matrix(data_train[particao$Resample1,])
  validation <- as.matrix(data_train[- particao$Resample1,])

  x_train <- as.matrix(train[, 2:(ncol(train)-1)])
  y_train <- as.matrix(train[, ncol(train)])
  x_validation <- as.matrix(validation[, 2:(ncol(train)-1)])
  y_validation <- as.matrix(validation[, ncol(train)])

  # Escalonando os valores dos atributos para que fiquem restritos entre 0 e 1
  x_all <- rbind(x_train, x_validation)
  x_all <- normalizeData(x_all, type = "norm")
  x_train <- x_all[1:nrow(x_train), ]
  x_validation <- x_all[(nrow(x_train)+1):(nrow(x_train)+nrow(x_validation)), ]


  length_train <- length(y_train)
  length_validation <- length(y_validation)
  for (p in ps){
    # Criando modelo:
    model <- mlp(x_train, y_train, size = p, maxit = 5000, initFunc = "Randomize_Weights",
                 initFuncParams = c(-0.3, 0.3), learnFunc = "Rprop",
                 learnFuncParams = c(0.1, 0.3, 0.1), updateFunc = "Topological_Order",
                 updateFuncParams = 0.1, hiddenActFunc = "Act_TanH",
                 shufflePatterns = TRUE, linOut = FALSE, inputsTest = NULL,
                 targetsTest = NULL)
    plotIterativeError(model)

    # Calculando acuracia de treinamento
    #y_hat_train <- predict(model, as.matrix(x_train))
    #yt <- (1*(y_hat_train >= 0.5)-0.5)*2
    #accuracy_train<-((sum(abs(yt + y_train)))/2)/length_train
    #print(paste("Acuracia de treinamento para p = ", p, "     ", accuracy_train))

    # Rodando dados de teste:
    y_hat_test <- predict(model, as.matrix(x_validation))
    plotRegressionError(y_hat_test, y_validation)
    yt <- (1*(y_hat_test >= 0.5)-0.5)*2
    accuracy_validation<-((sum(abs(yt + y_validation)))/2)/length_validation
```

```
    results[index, match(p, ps)] <- accuracy_validation
    print(paste("Acuracia de teste para p = ", p, " e ", accuracy_validation))
  }
}
print("
    ------------------------------------------------------------------------------------------------------------------------
    ")

for (p in ps){
  print(paste("Acuracia de teste media para p = ", p, " e ", mean(results[, match(p, ps)])
      ))
}
```

Listing 17: Algoritmo para validação da rede MLP em R

4.17 Validação da rede MLP - Keras

```
# mlp for multi-label classification
from numpy import mean
from numpy import std
from sklearn.model_selection import RepeatedKFold
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import accuracy_score
import pandas as pd
from keras.utils.vis_utils import plot_model
from sklearn import preprocessing



# get the dataset
def get_dataset():
    data = pd.read_csv("treino.csv")
    y = pd.DataFrame(data["y"], index=range(0,3176), columns=["y"])
    x = data.drop(columns=["y", "id"])
    scaler = preprocessing.MinMaxScaler()
    x_scaled = scaler.fit_transform(x.values)
    #x_new = SelectKBest(chi2, k=15).fit_transform(x_scaled, y)
    #print(x_new.shape)
    return x_scaled, y.values


# get the model
def get_model(n_inputs, n_outputs):
    model = Sequential()
    #model.add(Dense(10, input_dim=n_inputs, kernel_initializer='uniform', activation='
        selu'))
    model.add(Dense(35, input_dim=n_inputs, kernel_initializer='uniform', activation='tanh
        ', use_bias=True))
    model.add(Dense(n_outputs, activation='tanh', use_bias=True))
    #opt = keras.optimizers.SGD(
    #    learning_rate=0.01, momentum=0.05, nesterov=False, name="SGD")
    model.compile(loss='mean_squared_error', optimizer="Adam")
    plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
    return model


# evaluate a model using repeated k-fold cross-validation
```

```
def evaluate_model(X, y):
    results = list()
    n_inputs, n_outputs = X.shape[1], y.shape[1]
    # define evaluation procedure
    cv = RepeatedKFold(n_splits=10, n_repeats=1, random_state=1)
    # enumerate folds
    for train_ix, test_ix in cv.split(X):
        # prepare data
        X_train, X_test = X[train_ix], X[test_ix]
        y_train, y_test = y[train_ix], y[test_ix]
        # define model
        model = get_model(n_inputs, n_outputs)
        # fit model
        history = model.fit(X_train, y_train, verbose=0, epochs=3000)
        # make a prediction on the test set
        yhat = model.predict(X_test)
        yt = (1 * (yhat >= 0) - 0.5) * 2
        # round probabilities to class labels
        yhat = yt.round()
        # calculate accuracy
        acc = accuracy_score(y_test, yhat)
        # store result
        print('>%.3f' % acc)
        results.append(acc)
    return results

# load dataset
X, Y = get_dataset()
x = X
y = Y
results = evaluate_model(x, y)
# summarize performance
print('Accuracy: %.3f (%.3f)' % (mean(results), std(results)))
```

Listing 18: Algoritmo para validação da rede MLP em Python